Karlsruher Institut für Technologie

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung
Prof. Dr. rer. nat. Wolfgang Karl

Zentralübung Rechnerstrukturen: Vektorrechner

8. Übungsblatt – Musterlösung

1 Vektorrechner

a) DAXPY: $Y = a \cdot X + Y$ berechnet in Double-Precision.

```
LV       V1, Rx      ; load vector X
MULVS.D  V2, V1, F0  ; vector-scalar multiply
LV       V3, Ry      ; load vector Y
ADDV.D   V4, V2, V3  ; vector add
SV       Ry, V4      ; store result vector
```

Die Schleife, die nötig ist, um über die Vektoren zu iterieren, entfällt im Fall der Vektorrechnung.

b) Convoys:

a) `LV V1, Rx`

b) `MULVS.D V2, V1, F0`, `LV V3, Ry`

c) `ADDV.D V4, V2, V3`

d) `SV Ry, V4`

Folglich werden 4 Convoys benötigt. Da jeder dieser Convoys nach der Aufgabenstellung ein chime zur Ausführung benötigt, braucht man auch 4 chimes.

Zusammen mit 2 FLOP pro Ergebnis ergibt sich damit eine Rate von $\frac{chimes}{FLOP}$ von 2.

c)

Convoy	Startzeit	Erstes Ergebnis	Letztes Ergebnis
1. `LV`	0	12	$11+n$
2. `MULVS, LV`	$12+n$	$12+n+12$	$23+2n$
3. `ADDV`	$24+2n$	$24+2n+6$	$29+3n$
4. `SV`	$30+3n$	$30+3n+12$	$41+4n$

Die Zeit pro Ergebnis ist für einen Vektor der Länge 64:
$\frac{41+4*64}{64} = 4+(41/64) = 4,64$ Zyklen. Verglichen mit der Schätzung von 4 chimes ergibt sich, dass die genauere Rechnung aufgrund des Overheads für das Aufsetzen der jeweiligen Operationen um $\frac{4,64}{4} = 1,16$ mal höher ausfällt.

d) Stride von 1:
Latenz von 12 Zyklen für erstes Element und 63 Zyklen für alle weiteren zu holenden Elemente, da bei 16 Speicherbänken jeder nächste Zugriff auf die jeweils nächste Bank geht und somit die Latenz von 12 Takten versteckt werden kann. Folglich benötigt man 75 Zyklen oder 1,17 Takte pro Element.

Stride von 32:
Da 32 ein Vielfaches von 16 (der Anzahl der Bänke) ist, handelt es sich hier um den schlechtesten Fall. Jeder Zugriff des Strides geht auf die gleiche Memory Bank und kollidiert mit dem vorhergehenden. Damit benötigt jeder Speicherzugriff die Latenz von 12 Takten und man benötigt insgesamt: $12*64 = 768$ Takte oder 12 Takte pro Element.

e) Ohne Verkettung:

```
7 | 63 | 6 | 63 |
MULTV    ADDV
```

Gesamt 139 Takte

Mit Verkettung:

```
7 | 63 |
  | 6 | 63 |
```

Gesamt 76 Takte

$\text{Speedup}_{\text{Verkettung}} = \frac{139 \text{ Takte}}{76 \text{ Takte}} \approx 1,83$

Somit wird durch die Verkettung der zwei Operationen bei einer Vektorlänge von 64 Elementen bereits ein Speedup von $1,83$ erzielt.

f) Initialisierung:

```
MTC1 VLR, R1          # vector-length register := n
                      # wobei R1 den Wert n enthaelt
LV   V1, Ra           # int a[n] in V1 laden
LV   V2, Rb           # int b[n] in V2 laden
MOV  R1, 1            # R1 mit 1 initialisieren
CVI  V3, R1           # Create Vektor Index
                      # 0, R1 * 1, R1 * 2, ...
                      # entspricht for (i=0;i<n,i++)
```

Berechnung:

```
SLTV.D  V1, V2        # compare elements with 'Less Than'
                      # if true 1 in Vektor Mask Register
                      # else 0 in Vektor Mask Register
                      # if (a[i] < b[i])

SUBV.D  V2, V2, V2    # Komponenten von V2 mit 1
                      # im VMR werden auf 0 gesetzt

ADDV.D  V2, V2, V3    # diese Komponenten werden mit
                      # den Werten aus V3 aufgefuellt
                      # B[i] = i;
```

Abschluss:

```
CVM                   # Clear Vektor Mask
                      # alle Eintraege im VMR auf 1 setzen

SV  Rb, V2            # alle Komponent von V2
                      # an Adresse in Rb speichern
                      # entspricht Schreiben von b[i]
```